

Самомодифицирующиеся программы воспринимаются как нечто магическое, но при этом они весьма просты, и чтобы это продемонстрировать, я напишу такую программу под x86 архитектуру в NASM.

Базовый факториал

Для начала нам понадобится обычная программа вычисления факториала.

```
factorial:
    push ebp
    mov ebx, eax
factorial_start:
    sub ebx, 1
    cmp ebx, 0
    je factorial_end
    mul ebx
    jmp factorial_start
factorial_end:
    pop ebp
    ret
```

Здесь все довольно просто.

Самомодифицирующийся факториал

В алгоритме вычисления факториала есть два места, в которых изменение значения при выполнении имеет смысл: начальное значение и множитель.

Технические особенности

Во-первых, самомодифицирующиеся программы имеют свою специфику. По умолчанию nasm собирает программу без возможности ее дальнейшей самостоятельной модификации, потому что раздел .text из соображений безопасности отмечается как не перезаписываемый. Чтобы изменить флаги этого раздела для активации возможности записи потребуется задействовать objcopy и кастомную программу.

Мой скрипт для сборки этих программ лежит [здесь](#).

Начальное значение

В исходном коде начальное число передается через регистр eax. Чтобы использовать для этого самомодифицирующийся код, первым делом потребуется, чтобы в начале функции присутствовала обнуляющая инструкция mov для eax .

```
_start:
    mov dword [factorial+2], 0x5
```

```
call factorial
```

```
factorial:
```

```
    push ebp  
    mov eax, 0
```

Как видите, для передачи начального значения программа изменяет инструкцию `mov eax`. Значение 0 этой инструкции на 2 байта смещается от начала метода `factorial`.

Множитель

```
factorial_start:
```

```
    ; multiply  
    mov ebx, 0  
    mul ebx
```

Выше представлена заглушка, используемая для умножения. Далее нам нужна логика для установки `mov ebx, 0`, его декрементирования и выхода из цикла.

Инициализация множителя

Для установки множителя берем `ebx`, где хранится его первое значение, и копируем это значение в `mov eax, 0` в начало метода `factorial_start`.

```
factorial:
```

```
    ...  
    mov dword [factorial_start+1], ebx ; init decrementer
```

Декрементирование множителя

В стандартной программе логика будет такой:

```
    декрементировать множитель;  
    если он окажется 0, выйти;  
    перепрыгнуть назад.
```

В нашей самомодифицирующейся программе изменяется единственная деталь – декрементирование множителя.

Для этого необходимо получить его текущее значение, уменьшить это значение и скопировать обратно.

```
factorial_start:
```

```
    ...  
    ; decrement
```

```
mov ebx, dword [factorial_start+1]
sub ebx, 1
mov dword [factorial_start+1], ebx
```

Результат

Совмещая все это, получаем:

```
extern printf
```

```
section .data
    format: db "num: %d",10,0
```

```
section .text
    global _start
```

```
_start:
    mov dword [factorial+2], 0x5 ; start number
```

```
    call factorial
    ; print result
    push eax
    push format
    call printf
    ; exit
    mov eax, 1
    mov ebx, 0
    int 80h
```

```
factorial:
    push ebp
    mov eax, 0

    mov ebx, eax
    sub ebx, 1
    mov dword [factorial_start+1], ebx ; init decrementer
    mov ebx, 0
```

```
factorial_start:
    ; multiply
    mov ebx, 0
    mul ebx

    ; decrement
    mov ebx, dword [factorial_start+1]
    sub ebx, 1
```

```
mov dword [factorial_start+1], ebx
; exit if at 0
; could exit at 1, but then it doesn't handle 0x2
cmp ebx, 0
je factorial_end
; loop back
jmp factorial_start
```

```
factorial_end:
    pop ebp
    ret
```

Заключение

Я нахожу саомодифицирующиеся программы довольно интересными – их код выглядит несколько иначе, немного беспорядочен и содержит пустые значения, но при этом продумывать для них логику очень увлекательно.

Применяются они в различных областях, в основном относящихся к обфускации – к примеру, при реализации защиты лицензий или вредоносного ПО. Я подумываю создать на этом принципе собственный упаковщик или, по меньшей мере, прикольный crackme.

Если вам интересно познакомиться с другими примерами саомодифицирующихся программ под x86, то милости прошу в мой репозиторий.